Aalto University
School of Science
Master's Programme in Information Networks

Lauri Lavanti

Evaluating and designing an ontology editor for domain experts

Master's Thesis
Espoo, November 27, 2017

Supervisor: Professor Kari Smolander
Advisor: Katariina Nyberg M.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in Information Networks

ABSTRACT OF
MASTER'S THESIS

Author: Lauri Lavanti

Title:
Evaluating and designing an ontology editor for domain experts

Date:	November 27, 2017	**Pages:**	75
Major:	Information Networks	**Code:**	SCI3047
Supervisor:	Professor Kari Smolander		
Advisor:	Katariina Nyberg M.Sc. (Tech.)		

Zalando is building a Fashion Knowledge Graph and in order to populate the graph, input is needed from fashion experts, who are not experts in semantic web. While there are tools for managing semantic web and ontologies, concerns were raised about whether those tools are suitable for fashion experts, as they are usually aimed at ontology experts.

The research into this topic started with the definition of requirements for the tool necessary for this use case. With two given use cases as a basis, two workshops were held with the ontology engineering team and some representatives of end users at Zalando. Based on the results of these workshops a set of requirements were formalized and prioritized.

Based on the set of requirements the three selected tools, Protégé, TopBraid Composer and WebVOWL, were evaluated. After the evaluation it was clear that none of the three selected tools would fulfill the requirements and thus two prototypes were created to fulfill the requirements. The first prototype was based on an incomplete set of requirements and was a stand-alone prototype, while the second prototype was an extension on WebVOWL, that fulfills all the requirements. As a conclusion the recommendation is to implement the second prototype, while iterating the design with UX experts.

Keywords: ontology, semantic web, Protégé, TopBraid Composer, WebVOWL, VOWL, user experience, user interface

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Information Networks

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Lauri Lavanti		
Työn nimi:	Ontologian hallintatyökalun arviointi ja suunnittelu alan asiantuntijoille		
Päiväys:	27. marraskuuta 2017	**Sivumäärä:**	75
Pääaine:	Information Networks	**Koodi:**	SCI3047
Valvoja:	Professori Kari Smolander		
Ohjaaja:	Diplomi-insinööri Katariina Nyberg		

Zalando rakentaa muodin tietograafia ja jotta graafiin saadaan sisältöä, tarvitaan muotieksperttien panosta, jotka eivät ole eksperttejä semanttisessa internetissä. Vaikka semanttisen internetin ja ontologioiden hallintaan on olemassa työkaluja, niiden sopivuudesta muotieksperteille nousi huolenaiheita, sillä ne ovat tavallisesti suunniteltu ontologiaeksperteille.

Tutkimus tähän aiheeseen aloitettiin määrittelemällä vaatimukset tähän käyttötapaukseen sopivalle työkalulle. Käyttäen kahta annettua käyttötapausta pohjana, pidettiin kaksi työpajaa ontologia tekniikka ryhmän ja loppukäyttäjien edustajien kanssa Zalandolla. Työpajan tulosten perusteella muotoiltiin lista vaatimuksia, jotka järjestettiin tärkeyden mukaan.

Vaatimusten perusteella arvioitiin kolme valittua työkalua, Protégé, TopBraid Composer ja WebVOWL. Arvioinnin jälkeen oli selvää, ettei yksikään kolmesta työkalusta täytä vaatimuksia ja siksi tehtiin kaksi prototyyppiä täyttämään vaatimukset. Ensimmäinen prototyyppi perustui keskeneräiseen listaan vaatimuksia ja oli erillinen prototyyppi, kun taas toinen prototyyppi oli laajennus WebVOWLiin, joka täyttää kaikki vaatimukset. Lopputuloksena suositus on toteuttaa toinen prototyyppi iteroiden käyttäjäkokemuseksperttien kanssa.

Asiasanat:	ontologia, semanttinen internet, Protégé, TopBraid Composer, WebVOWL, VOWL, käyttäjäkokemus, käyttöliittymä
Kieli:	Englanti

Abbreviations and Acronyms

UI	User interface
UX	User experience
W3C	World Wide Web Consortium
RDF	Resource Description Framework; A W3C standard for defining linked data
SPARQL	SPARQL Protocol and RDF Query Language; A W3C standard for querying RDF
RE	Requirements Engineering; Methodologies for defining the requirements of a project
URI	Uniform Resource Identifier; A standard for defining things on the internet
IRI	Internationalized Resource Identifier; An extension of URI
JSON	JavaScript Object Notation; An open-standard file format
JSON-LD	A JSON-based Serialization for Linked Data; A W3C standard for defining the semantic web using JSON
XML	Extensible Markup Language; A W3C standard for defining data
RDFS	Resource Description Framework Schema; An extension of the RDF standard
OWL	W3C OWL 2 Web Ontology Language; A W3C standard for defining ontologies, extends RDF and RDFS
FOAF	Friend Of A Friend; An ontology for defining social and other types of relations
DC	Dublin Core; An ontology for defining geographical and other common physical attributes of locations and things
VOWL	Visual Notation for OWL Ontologies; A standard for the visualization of OWL ontologies

RDF/XML	A syntax for expressing RDF graphs in XML; A W3C standard
Turtle	Terse RDF Triple Language; A W3C standard for expressing RDF graphs
N-Triples	A line-based syntax by W3C for expressing RDF graphs

Contents

Chapter 1

Introduction

1.1 Motivation

Semantic web is a concept to model the internet as machine-readable. The way a human knows that both dogs and cats are animals, has to be taught to computers. Describing things in a machine-readable format is nothing new, but semantic web takes it a step further and instead focuses on the relationships between things and the kinds of relationships. This allows for the computer to infer additional information from data that is not explicitly stated, such as ancestors. Semantic web can be used to improve search results, display personalized content on websites for example and even Google is beginning to rely on semantic web to improve their search results [10].

Zalando SE (Zalando from here on out) is a German e-commerce company, specializing in fashion, founded in 2008 as an online shoe store, which was listed in the Frankfurt Stock Exchange in 2014 [23]. While Zalando only has a market share of one percent of the European fashion market [26], it has a yearly growth target of 20 percent and has so far been able to meet its targets. In 2013 Zalando decided to emulate the platform approach of the Chinese fashion market and began to build its own fashion e-commerce platform [24]. As a part of that platform Zalando is aiming to build a knowledge graph to standardize and centralize all of their product data.

1.2 Problem identification

Zalando is participating in building a semantic web by creating a Fashion Knowledge Graph to enhance their product data and further enable content customization and improve search results. In order to model a sufficient part of the fashion realm in semantic web it is necessary to have the fashion experts

within the company give their input directly to the knowledge graph. The main problem with this approach is that the fashion experts are not experts in semantic web and would thus require an as intuitive UI as possible. Thus the hypothesis for this thesis is *"None of the current ontology management tools are sufficiently intuitive for non-technical domain experts to use"*. Based on that the two research questions are as follows:

1. Are any of the existing ontology editors intuitive enough for users who are not proficient in semantic web?

2. If not, what should an ontology editor aimed at users who are not proficient in semantic web look like?

Based on the hypothesis the first step is to go through existing research into the subject and in accordance with the findings from them to build a testing framework for the UX of an ontology management tool. This is going to be extended with requirements defined with both technical and non-technical users of the system. With these requirements and a formal way of testing them, current solutions for ontology management will be evaluated. The assumption is that all of them will be found lacking in one area or another, but it is likely that some of them do other areas especially well and should thus be used as inspiration.

In case the assumption holds and the answer to the first research question is that none of the existing ontology editors are sufficient, the next step will be to design an ontology editor that is sufficient. The design will be iterative, with paper prototypes that fulfill all of the requirements defined in the workshops as the starting point. After designing the prototypes the same testing framework that was used on the existing tools will be run against the prototypes. Based on the results of those tests they will be further improved if necessary, otherwise they will be submitted as the recommended UI of the new editor.

On top of the research question, one more restriction is placed on the editor, which is that the fashion knowledge organization inside Zalando is building the ontology with a middle ware layer that uses SPARQL (more about SPARQL later) to query the knowledge. This means that the ontology editor must either use SPARQL itself or a middle ware must be built between the editor and the SPARQL endpoint to enable ontology management.

Chapter 2

Methodology

2.1 Process

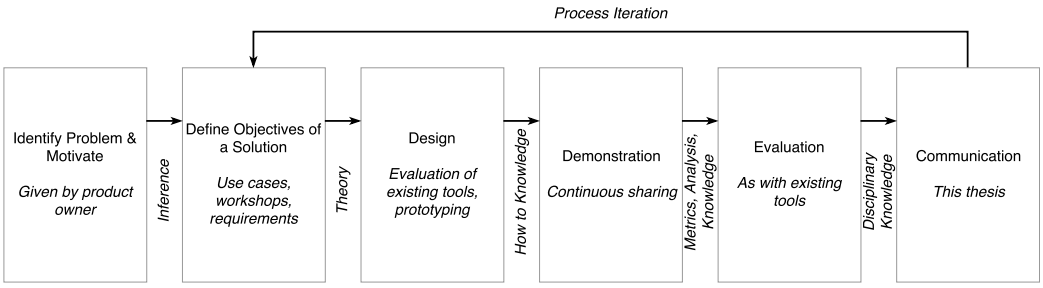

Figure 2.1: The process for this thesis work.

The research work will mainly follow the methodology defined by Peffers & al. [22]. The methodology is divided into six activities: problem identification and motivation, defining the objectives for a solution, design and development, demonstration, evaluation and communication. Due to the limited scope of the thesis the development part will be left out and instead most of the focus will be on defining the objectives and designing the solution.

The first activity has mainly already been done by the product owner at Zalando, and for the thesis most work on this will be conversations with the product owner and other involved parties. On top of that the product owner and other team members have already defined two use cases for the requirement which will be gone through in greater detail below.

The second activity will follow methodologies from RE. First the requirements will be defined in workshops and then refined with other methods de-

fined later. After the objectives have been clearly defined, existing solutions will be evaluated based on two methodologies, the UX Expert evaluation [25] and a property checklist [15].

The third activity will focus on the design part, as the development work was deemed out of scope for this work. Based on the list of requirements I will build a set of paper prototypes which will then be evaluated by stakeholders.

Demonstration will be accomplished by continuous sharing of the current set of requirements and designs. When the designs have been finished I will hold a knowledge sharing session with the rest of the team where I present the final designs and requirements and the results of the evaluation of existing tools.

Evaluation will be done by running the same set of techniques and tests that were run on existing tools against the prototypes. Additionally feedback will be asked from all team members and related product owners.

Main communication will be this thesis work, which will document all decisions and reasoning behind those decisions.

2.2 Defining the objectives for a solution

A good beginning for this kind of work would be to define some core use cases to be used as a basis for everything else in the project. A use case in this case is a definition for the goal of the product or project, which defines the functionality expected of the product. Use cases work as a way to remind you of the main goal when defining the requirements for a product and when evaluating or designing.

Usually there are several ways to discover and define use cases, but in this case the product owner for the Fashion Knowledge Graph already defined two use cases, which are as follows:

Use case 1 Allow any fashion domain experts to define grouping of products based on predefined concepts on Zalando's product data. For example, combining the concepts of dark colors, shirts and yoga wear to create a generated collection of dark-colored shirts suitable for yoga.

Use case 2 Allow named fashion domain experts and named ontology engineers to model the groupings on ontology-level according to an agreed-upon approval process, such as the 4-eye principle.

2.2.1 Workshops

Workshops are a good way to gather input from a larger group of individuals related to the product. In this case they were used to gather input about requirements for the semantic web management tool. The workshops started with a brief explanation of the desired output and the agenda for the workshop itself. Next, participants were given post-it notes and markers and told that they should write as many requirements they can think of per post-it within a 20 minute period. After everyone had written all the requirements they could think of, they each presented their own to the others. After those requirements had been grouped, each participant was given a set number of points which they were to distribute to the requirements they felt were most important. Initial prioritization of requirements was done based on the points they received, with a check afterwards were everyone was given a chance to comment on the order.

2.2.2 Requirements

A requirement in this context is a statement that defines some form of measurable property for a product or a process that is a condition for acceptance. Having requirements formally specified allows them to be tested and it helps with identifying situations when they are not being fulfilled. As there are multiple types of requirements, such as system requirements, stakeholder requirements etc. this study is only going to focus mainly on stakeholder requirements, with some system requirements as defined in restrictions for the study. [8]

Each requirement will be written in a sentence depicting what part of the product should fulfill what kind of requirement. As simple sentences like that are not capable of conveying all important information, attributes will be attached to the requirements with the following information: source of the requirement, priority of the requirement, whether the requirement pertains to the knowledge graph, the ontology or both and a verification method for the requirement (if any). [8]

2.2.3 Requirements prioritization

Requirements defined in the process of the thesis will be prioritized using the MoSCoW method by Dai Clegg [6] in 1994. In the MoSCoW method requirements are categorized in to four different categories. The first category is "must", which contains all the requirements without which the product would have no value. The second category is "should", which contains the

requirements with most of the added value that do not fit in the "must"-category. The third one is could, which contains the requirements that add value, but not as much as the ones in the "should"-category. The last category is "won't", which contains all of the requirements deemed out of scope for a project either through risk or value. In order to categorize requirements you define some sort of risk and value for each requirement to be able to fit them in to the four categories. [3] When working in a group, the MoSCoW method can be implemented for example by giving each participant a limited amount of points with which to prioritize the requirements they feel should be the highest priority.

While MoSCoW is a rather well-defined and simple process of prioritizing requirements, other options are available. Some alternatives are: needs-based analysis, where you try and identify which requirements are truly needed and which ones are not; crowd sourcing, where you have a crowd defining and prioritizing the requirements; dot voting, where participants have three to five dots with which to vote on the features they believe to be most important and buy a feature, where each participants gets a given amount of money to spend on features (with each feature having a price) and then using the money to buy their dream product. [7] The workshop prioritization process in this case is actually a combination of dot voting and the MoSCoW method, as the MoSCoW method by itself does not include a set method for gaining input from stakeholders.

2.3 Design and evaluation

Based on the restrictions and requirements all existing tools will be evaluated using two methodologies. Firstly the UX Expert evaluation [25], where I evaluate the UX of a system without input from users. The second methodology is doing a property checklist [15] for all the requirements defined for the product, allowing matching of all alternatives to the necessary requirements.

The first method of evaluation is the UX Expert evaluation. In UX Expert evaluation an expert goes through the system being evaluated and considers weaknesses and strengths of the system in terms of UX. While a cheap and fast method, it is not very accurate or scientific as it relies on the expertise of the person doing the evaluation. Nonetheless, it is a good way to get a starting point in comparing different tools. [25]

The second methodology is a property checklist. In a property checklist an expert goes through a certain list of properties or requirements and checks whether a tool meets those criteria. In this case the list of properties to go through will be the list of requirements. As property checklist is a systematic

way to go through even a long list, it does not matter if there are a lot of requirements defined for the tool. [15]

After the requirements have been defined for the ontology editor to be designed, multiple rounds of prototypes will be built. Prototypes are a cheap way for validating designs and the prototypes built in this case will be evaluated with key stakeholders and team members. In accordance with the agile mindset, the prototypes will be iterated based on user feedback and the evolution of requirements. Prototypes are a good match to the iterative process, as they are cost-effective and quick to make. As the design files will be kept during the process, small changes are simple and easy to make as well. As an alternative to prototyping, the product itself could be built, with a limited set of features developed with dummy data or end-to-end functionality. Unfortunately as it is a technically challenging project and the backend services do not exist yet, this alternative method was deemed as too expensive.

The prototypes will be evaluated with the same testing frameworks as the existing editors. Starting with the expert evaluation, which will be given less weight with the prototypes as the prototypes themselves will be designed by the same person as the one performing the evaluation and ending with the property checklist, where the prototypes will be evaluated against the set of requirements.

Chapter 3

Semantic web and tools for its management

3.1 Semantic web

The purpose of semantic web is to make the information on the internet accessible and usable to computers, allowing better personalization, linking and combining data among other benefits [1]. Semantic web can also be described using the five Linked Open Data principles by Tim Berners-Lee [2]:

1. Make your data available on the web with an open license
2. Make your data machine-readable (a table instead of a picture of a table)
3. Make your data available in a non-proprietary format
4. Use open standards from W3C, such as Resource Description Framework (RDF), which will be described in detail later
5. Link your data to other people's data

In practice the most important thing in semantic web is to give things unique identifiers as (dereferenceable) URIs [19]. IRIs can be used in place of URIs, as they are an extension to the URI specification with the difference that IRIs allow for international characters, such as Chinese or Arabic [9, 19]. URIs allow consumers to link and explore said things by giving them a link to follow for more information and related entities. The second most important characteristic of semantic web is to format data using a W3C standard, such

as RDF or SPARQL. RDF has several syntaxes and extensions, of which in the case of web applications the most intuitive is JSON-LD as JavaScript in browsers natively handles JSON, which was built for JavaScript-applications. [12]

Even with the improvements in artificial intelligence (AI) and machine learning (ML) a computer still has a hard time differentiating between two words, other than the differences in characters. Semantic web gives those words meaning, by strictly defining relationships and properties for words, thus allowing the computer to reason between connections and differences in words.

Semantic Web is built on RDF, which was proposed by Ora Lassila and Ralph R. Swick in 1999 [16] as a format of meta data to make the World Wide Web machine-understandable. The RDF syntax was originally built on top of XML [16], but the model is not dependant on XML and has later been extended to other syntaxes such as JSON. RDF was later extended with RDFS, which defines vocabulary for modelling data in RDF, by introducing semantic extensions on top of RDF [4]. RDF and RDFS are further extended by OWL, which is a language for expressing ontologies. OWL defines several modelling features for data, including ways to define relationships between classes, which can then used for reasoning about the data (determining implicit relations with data) [13].

While RDF, RDFS and OWL allow defining data that is understandable for machines and reasonable it can be taken further by using well-defined existing ontologies built on top of the RDF specification. One of those ontologies is FOAF, which defines properties and characteristics for social networks and people, such as "foaf:familyName" and "foaf:givenName". FOAF also defines similar properties for the social web [5]. Another example of such an ontology is the DC, which defines common terminologies to describe resources both on the internet and in the physical realm, such as books and CDs [5, 14]. In addition to FOAF and DC, one of the most common ontologies is DBpedia, which is a community-driven project to translate data from Wikipedia into an ontology [17]. The Zalando fashion ontology is aimed at becoming the definitive ontology for the fashion realm, like FOAF is for social networks. Zalando is also planning to take advantage of `schema.org`, which is a community-driven hosting of different schemas (ontologies), backed by companies like Google, Microsoft, Yahoo and Yandex. The ultimate goal of `schema.org` is to have a central location for schemas for structured data on the internet and beyond. [11]

The current plan of work on semantic web in the context of Zalando is built on three levels. The first level is to intake the current product data at Zalando and transform it into graph form. The resulting triples will be

part of the Fashion Knowledge Graph, within the name space "zk". The second level is to manually curate the information in the first level into a more efficient model. The intention is to remove duplicates and improve data quality. The second level will also be a part of the Fashion Knowledge Graph, within the name space "zk". The third level is the distilled universal fashion knowledge that is linked to equivalent concepts within the first level. The resulting graph will be the fashion ontology, within the name space "zo", aimed to become the de facto ontology in the fashion realm.

3.2 Existing editors

There has been a plethora of different editors and management tools for semantic web over the years, but lately most of them have been discontinued or their development has been halted. As it is, only two proper editors are still actively maintained, Protégé by Stanford University [20] and TopBraid Composer by TopQuadrant. Additionally there exists a tool to view semantic web graphs called WebVOWL made by the Visual Data Web project [18], which has contributions from multiple researchers from different universities.

As the selection of editors is so limited Protégé has been chosen as a representative of open sourced editors, while TopBraid Composer has been chosen as a representative of commercial editors, because both of them are the most maintained editors. WebVOWL is included in the evaluation as even though it is not an editor it is a very recent addition in the field of semantic web tools and is highly extensible due to being open sourced and developed with the latest technologies.

Chapter 4

Defining the objectives for a solution

4.1 User definitions

There are two types of users in the context of this thesis. The first type of users are fashion experts, who are experts in the domain of fashion, but are not necessarily technical experts or have prior knowledge about semantic web and ontologies. The second type of users are ontology engineers, who are technical experts in the domain of semantic web and ontologies. The assumption is that the fashion knowledge graph and the fashion ontology get most of their content from the fashion experts, while the ontology engineers ensure that all content in the graph and the ontology follow the rules of semantic web, while remaining efficient and logical.

4.2 User stories

In order to have valuable and relevant requirements to evaluate the tools against, two workshops were held to get input from both the developers of the Fashion Knowledge Graph and representatives of the non-technical users. Even though requirements could be inferred from the given use cases, having input from actual users and developers can help with identifying requirements you have not thought of and in prioritizing those requirements. In order to minimize missed requirements two workshops were held with the first set of requirements as a basis for the second one.

4.2.1 First workshop

Defining user stories started with a workshop with the entire ontology engineering team at Zalando, at the time approximately ten people, and the corresponding product owner, who is counted a fashion expert in the context of this work. The product owner ran the first session. First she described the current situation to the team, that we are planning on building our own ontology editor (or finding an existing one) and for that we need to understand all the requirements we might have for the editor. After the explanation we gave everyone a stack of post-it notes and then had twenty minutes of brainstorming where everyone filled as many post-its as they could, with one feature or requirement they thought necessary or useful per post-it. After filling their post-its each member took turns in putting them on the wall and explaining what their post-its contained. When everyone had put their post-its on the wall we grouped them together based on themes and then ended the session.

After the workshop I took the requirements written by team members and first removed duplicates. Next I rewrote the remaining requirements into user stories, with an active form and activity. I also added additional information to the user stories, giving them an ID, recording the source they came from, which graph they relate to (ontology, knowledge graph or both) and a way to verify that the requirement has been fulfilled. Finally I added priorities to the requirements using the MoSCoW method, created by Dai Clegg [6] in 1994.

Requirements that received the "must" priority were: creating and editing triples; peer-review of changes; having human-readable IRIs for entities; editing and creating entities using a form and having a visual representation of the graph. The requirements with the "should" priority were: using external properties and entities; filtering and traversing of the visual representation; seeing relevant entities and visualizing hierarchies within the graph. The requirements within the "could" priority were: seeing deduced predicates; adding entities by selecting the location in graph; seeing example search results from the Zalando fashion store; seeing suggestions for new entities and exporting the graph as CSV or JSON. The exact listing of the requirements defined in the first workshop on 10.7.2017 can be found in table A.1

4.2.2 Second workshop

The main objective of the second workshop I ran was to refine the previously defined requirements and discover possible new ones. Invitation list included

all members of Zalando's ontology engineering team and the corresponding product owners. The workshop began with an explanation of the project (defining requirements for an ontology editor in the context of Zalando) and a walk-through of the previous workshop's results. After the explanation had concluded participants were given post-it notes and markers for the following brainstorming session. As the expectation was that most of the necessary requirements have already been defined, the part where everyone wrote down requirements or features they thought necessary was only 15 minutes, after which everyone took turns in putting their post-its on the wall next to the old requirements and explained what they had written down. Next up was a prioritization exercise, where each participant was given three points to distribute per priority (Must, Should, Could, Won't). Lastly a short discussion was held on the topic, to ensure no misunderstandings were left.

Requirements that received the "must" priority were: creating and editing triples; peer-review of changes; viewing and filtering a visual representation of the graph; selecting the domain to manage; choosing between a visual and a tabular representation and creating entities with human-readable IRIs. The requirements with the "should" priority were: editing and creating entities using a form; seeing the inference layer of the graph; seeing the history of edits; seeing suggestions for new predicates; traversing the visual representation; seeing relevant entities; copying an existing entity and seeing examples of search results from the Zalando fashion store. The requirements within the "could" priority were: adding plugins to the editor; using external properties and entities; visualizing hierarchies within the graph; running the editor against any graph; assigning a review to specific people; seeing clusters of entities in the visualization; exporting the graph as JSON-LD or other similar format; importing a graph as JSON-LD or other similar format; adding entities by selecting the location in graph; seeing suggestions for new entities; marking an entity as "badly modelled" and seeing all reviews awaiting attention. The exact listing of the requirements defined in the second workshop on 26.9.2017 can be found in table A.2.

Chapter 5

Design

The first part of this chapter is a walk-through of the selected existing tools. The walk-through consists of a description of different (relevant) views of the editors, commentary on their strengths and weaknesses and finally a breakdown on how well they fulfill the requirements defined in chapter 4. The second part of this chapter is the representation and walk-through of the two prototypes created as a part of the research process. The first prototype is based on the first set of requirements, while the second one is intended as an extension on WebVOWL that fulfills the final set of requirements.

5.1 Existing tools

The selected editors to evaluate in this part are WebVOWL, Protégé and TopBraid Composer. The main reason to select these three is that most other editors for ontologies and semantic web are either discontinued or have not been updated within the last two years, which was taken to mean that they are deprecated. The other reason is that the first two editors chosen (WebVOWL and Protégé) are open sourced, thus allowing further extension and additional features, while TopBraid Composer is a commercial product, giving a good comparison point between open sourced products and commercial ones.

In order to do a property checklist against existing tools, the list of all requirements is too long and thus I have condensed them into six categories, which can be seen in table B.1. When doing the property checklist the tools will be evaluated against these six categories, with four points given for each fulfilled "must" requirement, two points for each "should" requirement and one point for each "could" requirement. If a requirement is not fulfilled completely it will be given half of the maximum points for that spe-

cific requirement. The six categories chosen are: management, for the actual management of semantic web data; review, for reviewing changes; representation, for the user interface and its applications; ontological capabilities, for displaying and exploring ontological features of the graph in the editor; suggestions, for suggestions and information not explicitly in the graph; extensibility, for extensibility of the editor. Each of the categories has a separate amount of maximum points, based on the amount of requirements related to each category, with management having 20 maximum points, review and representation having 17 maximum points, ontological capabilities having 4 maximum points, suggestions having 7 maximum points and extensibility having 3 maximum points.

5.1.1 WebVOWL

VOWL is a specification for visualizing OWL-based ontologies [21]. VOWL uses graph-format to visualize ontologies, built on clearly defined visual notations for different types of elements within OWL ontologies. On top of the visual elements VOWL also defines rules for splitting elements to reduce clutter and improve readability. The default visualization of graphs in VOWL uses a force-directed layout, which displays all edges as equally long, while avoiding crossing edges, tending to focus the nodes with most edges to the center of the visualization. For different types of OWL elements, the specification defines both a geometrical shape and a colors, with the colors being defined both in hexadecimal (RGB colors) and purpose, allowing users of the specification to define their own colors while still having the relativity of colors defined. VOWL was specified based on user studies, comparing existing visualizations of ontologies. [18]

There are two main implementations of VOWL. The first is ProtégéVOWL, which is a plugin for Protégé to visualize the ontology being edited, following the VOWL specification. The second implementation of VOWL is WebVOWL, available at `http://vowl.visualdataweb.org/webvowl.html`, created by the developers of VOWL. WebVOWL follows the VOWL specification, while providing filters not in the specification. One such filter is of special interest, called the degree of collapsing, which removes classes from the graph that do not have enough nodes that they are connected to, allowing the user to limit the visualization to the most central nodes in the graph, reducing clutter in the view. WebVOWL is open sourced and available with the MIT license in GitHub at `https://github.com/VisualDataWeb/WebVOWL`. [18]

While not an ontology editor, WebVOWL is an excellent example of ontology visualization and the application is open source so extending it is possible. WebVOWL is an example implementation of the VOWL specifi-

cation made by the team that defined VOWL. WebVOWL is built on top of D3 (available at `https://d3js.org/`), a JavaScript library for visualizing data in the browser and is thus highly extensible. As visualization is one of the most technically challenging features required for the editor we choose, having a ready-made solution, such as WebVOWL, as a starting point could save a lot of time and resources.

The problem with WebVOWL is that it does not digest OWL as is, but requires everything to be run through a converter first to be able to access all functionalities. The converter is open source as well and available at `https://github.com/VisualDataWeb/OWL2VOWL`, but the requirement of having to use such a converter means that WebVOWL would not be usable with the current SPARQL implementation of Zalando.

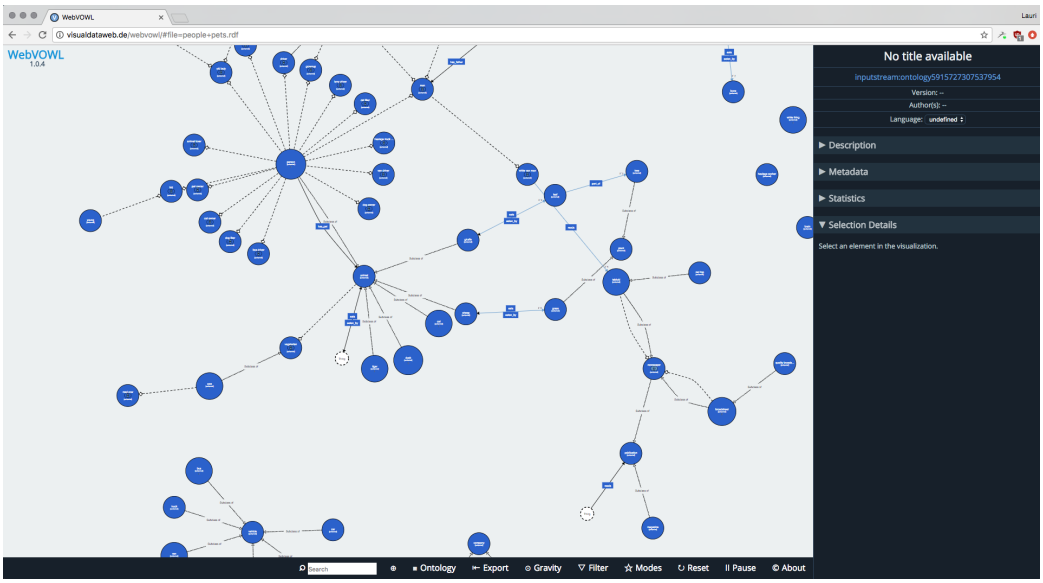

Figure 5.1: The main view of WebVOWL.

The main view of WebVOWL (as seen in picture 5.1) after importing the "People and Pets" ontology by the University of Manchester available at `http://owl.man.ac.uk/tutorial/` shows the given ontology as a graph. On the right there should be information about the ontology visible, but the example ontology does not have all of the data defined, so the information is rather minimal. On the bottom there are controls to interact with the graph in different ways, such as filtering, exporting and resetting the graph animation. As a view this is very informative as an overview of the ontology, but it would help to have more information present on the sidebar to the

right. Additionally you can drag and drop any node in the graph view and the rest of the graph will adjust accordingly, in case you feel the need to isolate something from the center of the graph for example.

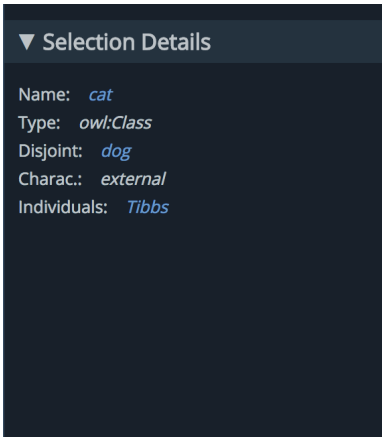

Figure 5.2: The entity view of WebVOWL.

After selecting an entity from the graph you get to view the details of an entity (as seen in picture 5.2) on the sidebar. The sidebar is separated into a description, meta data, statistics and details of the selected entity, where the details tab is the only one expanded by default. The details tab shows relations to other entities and other defining characteristics for the chosen entity, with links to their URLs. The statistics tab shows statistics about the ontology itself, not the entity, which is counter intuitive. In the case of the example ontology, the meta data and description tabs do not show anything for the selected entity.

The selection view is a perfect candidate for allowing the editing of chosen entities and showing relevant data. Although it should be more customized according to the chosen entity (for example it does not show the selected entity as title, but shows the title of the ontology), it is a good starting point for an entity view.

The search view of WebVOWL (as seen in picture 5.3) is a simple functionality to find entities in the graph. You start typing, it shows suggestions from the graph and clicking one leads the view to highlight the one you were searching for. This kind of search is helpful especially when the graph is bigger than the example one, although having the search filter the visible elements would be very useful as well.

After selecting a combination of filter properties from the bottom bar (as seen in picture 5.4), the graph view will filter non-relevant entities from the

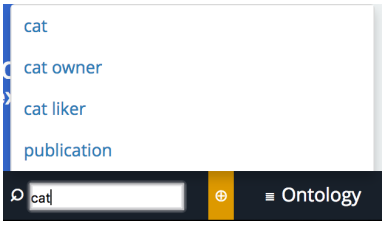

Figure 5.3: The search view of WebVOWL.

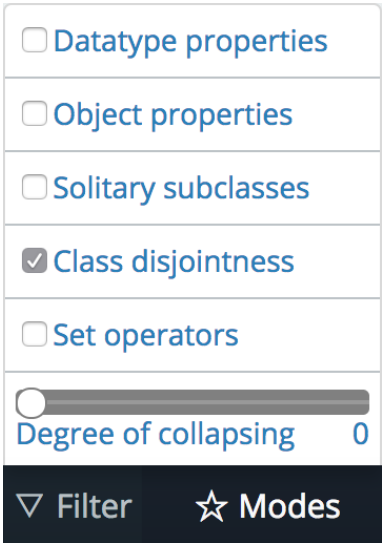

Figure 5.4: The filter view of WebVOWL.

visualization. While the filtering options in WebVOWL are probably very helpful for ontology engineers, but for fashion experts more simplistic and intuitive filters might be more useful.

The exact fulfillment of requirements for WebVOWL can be seen in the appendix table C.1, but the points per category can be seen in chart 5.5 as percentages. The absolute values are 9½/20 for management, 8½/17 for review, 12/17 for representation, 2½/4 for ontological capabilities, 3½/7 for suggestions and 2/3 for extensibility. The total points for WebVOWL are 38/68 points.

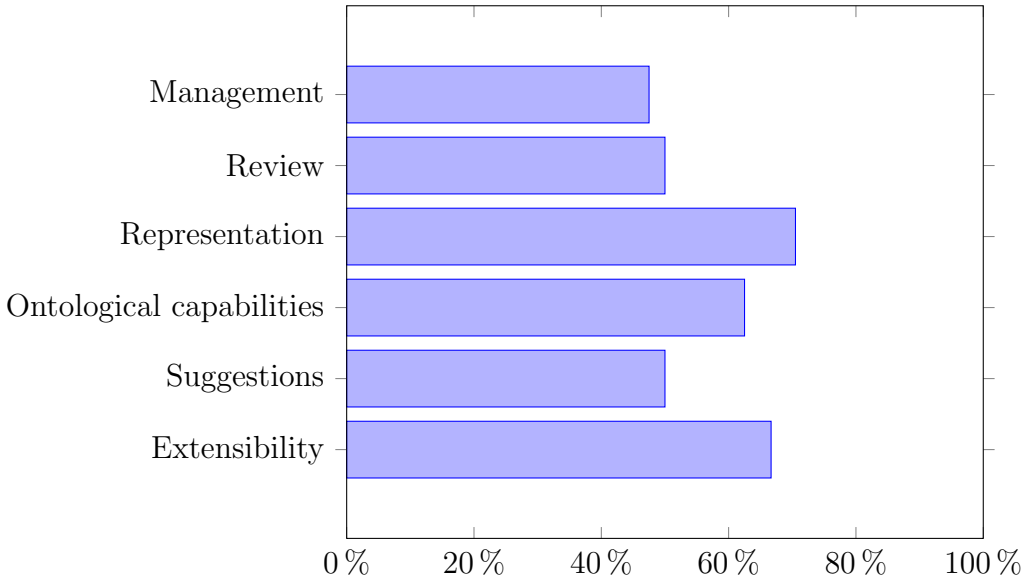

Figure 5.5: WebVOWL fulfillment of requirements

5.1.2 Protégé

Protégé is the most common ontology editor. It begun in the 1980s, and remains in active development to this day. It is being developed at Stanford, and is open source. Protégé has two main applications. Protégé 5, a desktop system, has advanced features and used to be the most used out of the two. The second one is WebProtégé, which is simpler than Protégé 5, but as it is a web application, requires no installation and is sufficient for most use cases. [20]

After downloading Protégé I imported the "People and Pets" ontology. After importing the ontology, the main view of Protégé does not clearly display the most important data for the use case, the entities and the individuals. Upon closer examination one can see that there are tabs for "Entities" and "Individuals by class", but that does not diminish the fact that the main view of a data editor should show the data in some format that is useful, whereas the only useful information the main view of Protégé displays are some metrics of the imported ontology.

The entities view of Protégé (as seen in picture 5.6) shows a list of different entities in the ontology, based on inheritance, on the left and the details of a selected class on the right (including instances of said class). The hierarchy visible on the left can be selected to be based on different types of properties, classes (the default), object properties, data properties et cetera. On the right

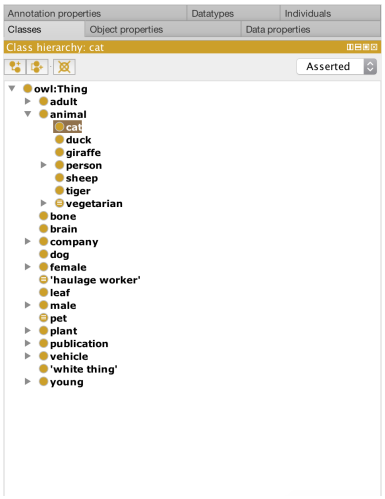

Figure 5.6: The entities view of Protégé.

(as seen in picture 5.7) you can select to view either the classes properties or the usage of the class (where the class is in use in the ontology). As a view this is a lot better than the main view and is more like what the main view should be like. Even though the view shows a good overview of the selected class and its properties there are no clear ways on how to add new classes or properties, though adding, editing and removing values for existing properties is simple and clear.

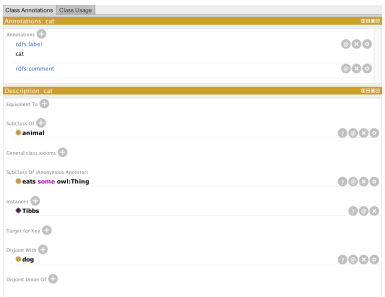

Figure 5.7: The entity view of Protégé.

An especially useful feature of the entities view is the class usage tab (as seen in picture 5.8). The tab details all classes that include the selected class in one of its properties. The tab also includes instances of the selected class, which should be its own tab though.

The individual view (as seen in picture 5.9) shows all relevant data of

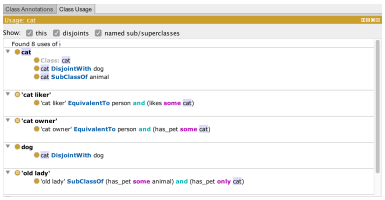

Figure 5.8: The class usage tab in entity view of Protégé.

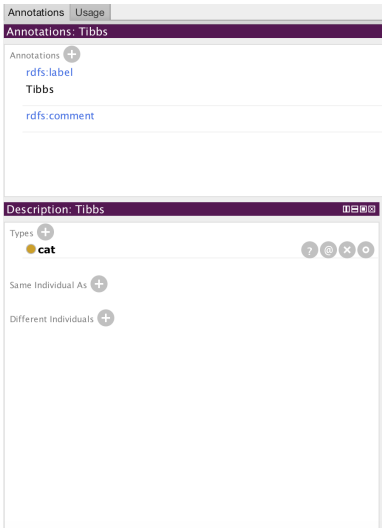

Figure 5.9: The individual view of Protégé.

an individual. The first step is choosing a class in the class hierarchy and then the instance you want to view or edit. At that point you see the details of the individual. The view is a bit cluttered as there are multiple tabs for an individual visible by default. Otherwise the view is very similar to the entities view and suits its purpose.

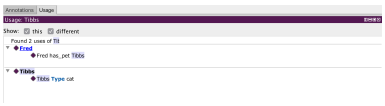

Figure 5.10: The usage tab in individuals by class view of Protégé.

Another tab in the individuals by class view is the usage tab (as seen in picture 5.10) which shows all usages of the selected entity. When dealing

with linked data that kind of a view is very useful. In this case though the tab is too small in case there are multiple links to the selected entity.

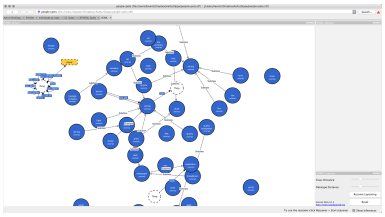

Figure 5.11: The VOWL view of Protégé.

There are many different plugins for visualizing ontologies in Protégé but for this review I am using the VOWL plugin for Protégé (as seen in picture 5.11) as it is being considered for the visual design language of the possible editor due to its simple and refined definition. The view is a rather simple graph representation of the entities in the ontology with the ability to drag nodes and select them. When a node is selected its information is shown on the right in the sidebar. Other than that when the animation is running you can edit the distance between classes and between data types. Otherwise the plugin is lacking in features. There is no way to view the instances of classes, the shown details are incomplete and if you drag one node none of the others react to the changed ordering unless the layout is being animated. Also the animation does not stop at any point.

The exact fulfillment of requirements for Protégé can be seen in the appendix table C.2, but the points per category can be seen in chart 5.12 as percentages. The absolute values are 18/20 for management, 8½/17 for review, 9½/17 for representation, 2½/4 for ontological capabilities, 3½/7 for suggestions and 3/3 for extensibility. The total points for Protégé are 45/68 points.

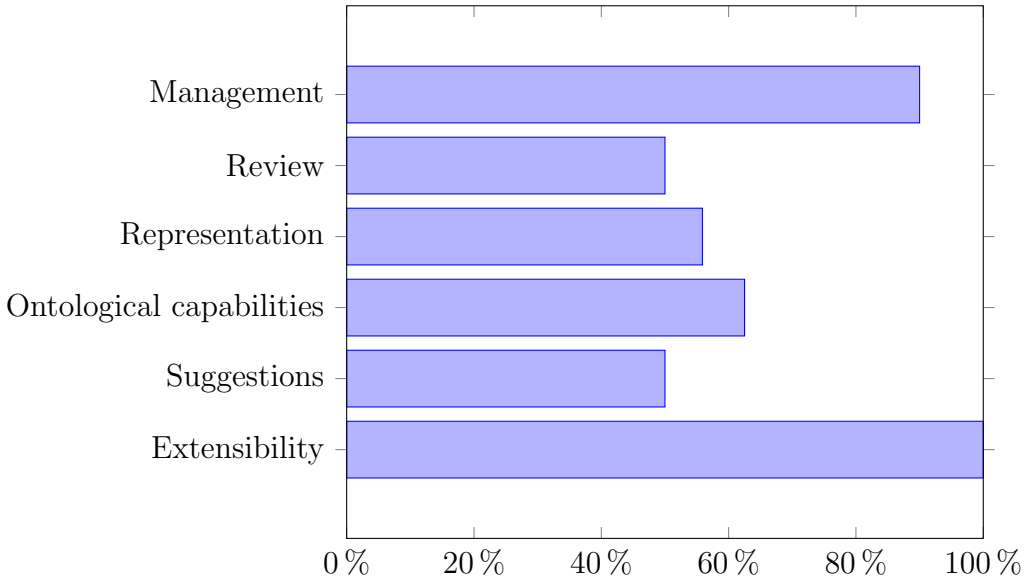

Figure 5.12: Protégé fulfillment of requirements

5.1.3 TopBraid Composer

As with Protégé the main view of TopBraid Composer is not very useful, except for the list of entities on the left. The main focus is on the information about imported ontology, which does not even show similar metrics as Protégé's main view. The main view should be relevant to the main use case, which it is not in this case. Another downside with the editor is the plethora of different buttons with no associated description of their functionality, which will only confuse the average user.

By selecting one of the entities from the entities view (as seen in picture 5.13) you get to see the details of a single entity (as seen in picture 5.14), The main view is a very traditional form to edit the chosen entity, with the existing details shown by default. The form is simpler than the one in Protégé but there are no significant differences. Unfortunately TopBraid Composer does not have a similar usage view as Protégé but it does have a source code view, which shows the selected entity as RDF/XML, Turtle, N-Triple or JSON-LD.

The instances view of TopBraid Composer (as seen in picture 5.15) is similar to the entities view, but unlike the class tree, instances can be seen in the bottom tab. Unlike in the entities view, the instance view (as seen in picture 5.16) shows incoming references in the editing form, but not as extensively as Protégé. Similar to the entities view, the instance view also

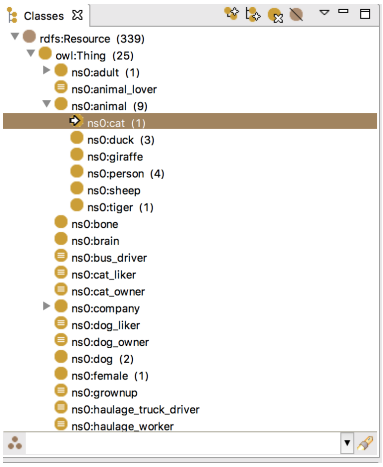

Figure 5.13: The entities view of TopBraid Composer.

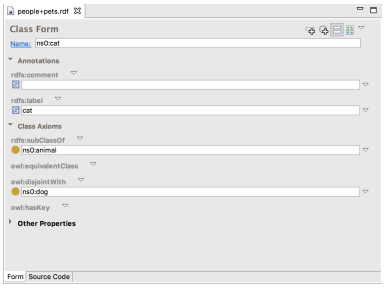

Figure 5.14: The entity view of TopBraid Composer.

has a source code tab that shows the instance in either RDF/XML, Turtle, N-Triple or JSON-LD as seen below.

Unfortunately as visualization and many other features are paid features of TopBraid Composer I did not get to evaluate them. In addition, unlike Protégé, TopBraid Composer is not open source and thus cannot be extended at will and the usability and features of TopBraid Composer are set.

The exact fulfillment of requirements for TopBraid Composer can be seen in the appendix table C.3, but the points per category can be seen in chart 5.17 as percentages. The absolute values are 15½/20 for management, 0/17 for review, 11½/17 for representation, 1½/4 for ontological capabilities, 0/7 for suggestions and 2/3 for extensibility. The total points for Protégé are 30/68 points.

Figure 5.15: The instances view of TopBraid Composer.

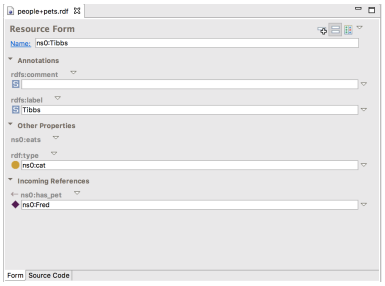

Figure 5.16: The instance view of TopBraid Composer.

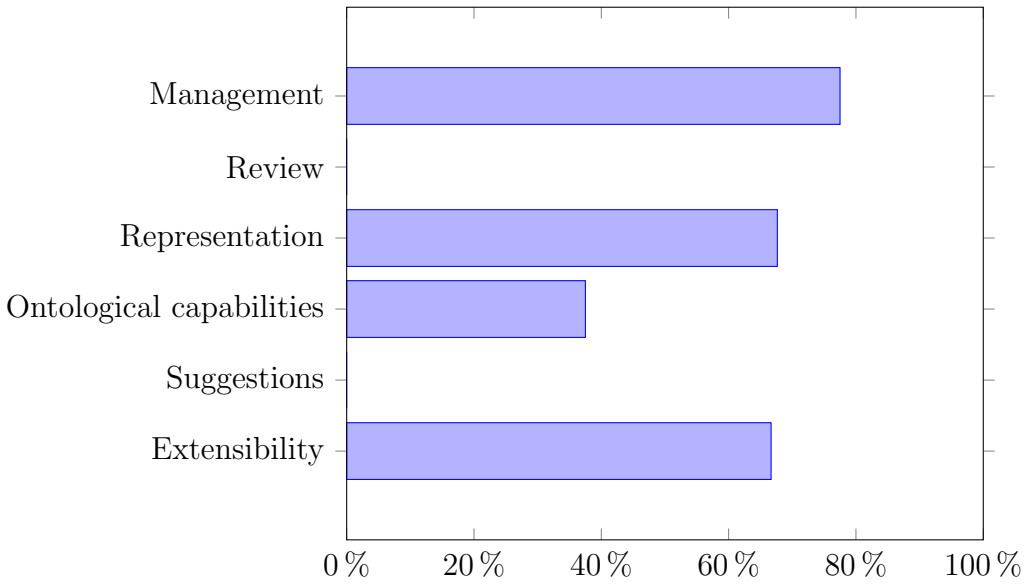

Figure 5.17: TopBraid Composer fulfillment of requirements

5.1.4 Conclusion

As can be seen from the division of points from figure 5.18, looking at purely the requirements means that Protégé is the top contender and TopBraid Composer has the least requirements fulfilled. The exact points are 38/68 points for WebVOWL, 45/68 points for Protégé and 30/68 points for TopBraid Composer. However, I think that it should be taken into account that in the case of WebVOWL, although it did not fulfill many requirements, the rest of them could be added by contributing to the open source project. As for TopBraid Composer it had mainly similar features as Protégé without plugins, but only in the commercial version and there is no way to extend it by yourself. As for the usability of these tools, WebVOWL was superior to both Protégé and TopBraid Composer, while Protégé and TopBraid Composer are very similar to each other, with Protégé having a slight advantage.

Overall the suggestion would be to either teach domain experts the use of Protégé or expand the features of WebVOWL. As both Protégé and WebVOWL would require development work to fulfill all the requirements I would highly recommend building on top of WebVOWL as it has better usability from the start and would thus fit better for the domain experts.

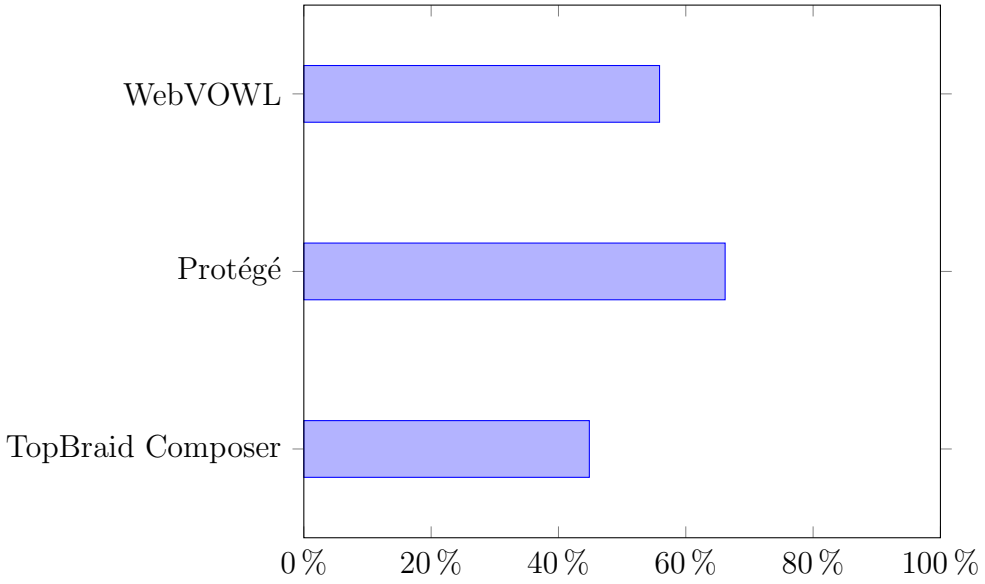

Figure 5.18: Fulfillment of requirements for all editors

5.2 Custom tool

5.2.1 First prototype

Based on the first set of requirements I created a prototype using MockPlus, available at `https://www.mockplus.com/`, that was aimed at creating a visual graph based editor and making sure all the requirements defined in the first phase are met by the prototype. The prototype itself is simply a collection of different views, but MockPlus could be used to make it into an interactive demo with limited capabilities. The visual design language is meant to follow the VOWL specification [21]. On the upper left corner of each view of the prototype are listed the requirements it fulfills, using the identification present in the first table of requirements. The code name for the tool being designed is "Bluecell" and from now onward "Bluecell Alpha" will be used to designate the first prototype.

The main view of Bluecell Alpha (seen in picture 5.19) shows the instances of entities in a graph-format (simplified here to save time when making the prototype). It fulfills the requirements *"I can view a visual representation of the graph"*, *"I can traverse the visual representation of the graph"* and *"I can export the graph as a CSV or a JSON"*. On the top part of the UI is a search bar and a filter drop-down, both of which filter the visible graph. On

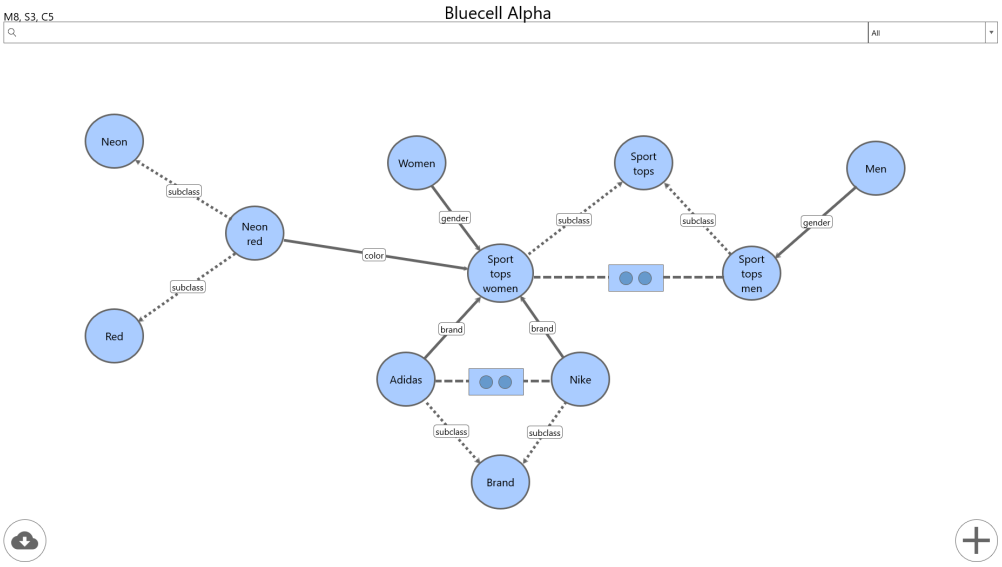

Figure 5.19: The main view of Bluecell Alpha.

the lower left corner is a button to download the graph as RDF, Turtle or JSON-LD. On the lower right corner there is a button to create a new entity.

What this prototype is missing, is a way to influence the entities themselves, but is instead focused simply on the instances. Also there is no way to select the graph being edited, so unlike Protégé and TopBraid Composer there is only one editable ontology. In practice this would be done by deploying the prototype to with different environment variables that would connect it to the correct ontology (either the instances or the entities themselves) and accessed through a browser.

After clicking the "create new instance" button, the corresponding view (as seen in picture 5.20) pops up and gives one a form with which to define the new instance. The form fulfills requirements *"I can create triples"*, *"I can edit triples"*, *"After modifying the ontology, my changes will not be visible until verified by a domain expert and an ontology engineer"*, *"After modifying the knowledge graph, my changes will not be visible until verified by a peer"*, *"IRIs for the entities I create may be human-readable"*, *"I can edit and create entities in the graph using a form with the labels of IRIs"*, *"I can use well-defined properties and entities, such as FOAF and DBpedia"* and *"I can see deduced predicates in entities"*. In the edit view the image has already been filled with the name, sub classes, labels and a description. When the sub classes and other relations have been defined it shows up in the graph. After defining the instance you can either submit it for review or cancel the changes

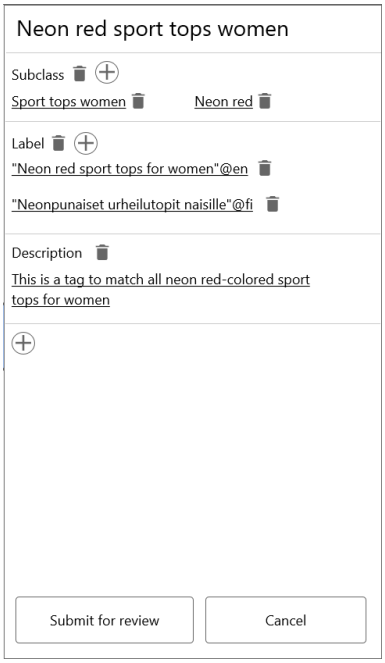

Figure 5.20: The editing and creating new view of Bluecell Alpha.

you have made. Some details have been overlooked in the prototype in favor of clarity.

The review changes made by others view (as seen in picture 5.21) is the functionality missing from both Protégé and TopBraid Composer. The review view fulfills the requirement *"I am able to review changes to the graph made by others"*. The view shows the changes being reviewed both in the graph and in the form, with the form showing the details of the proposed changes. After reviewing changes you can type in a comment and either accept the changes or request for new changes. As per before some details have been overlooked for simplicity.

The filter view of Bluecell Alpha (as seen in picture 5.22) is reached by either selecting a property from the top-right drop-down list or typing something in the search field in the top. It fulfills the requirements *"I can filter the entities visible in the visual representation of the graph"* and *"I can visualize hierarchies within the graph based on transitive predicates"*. When you select one of the properties from the drop-down menu it only shows relations of that property (as seen in the picture). When you type something into the search box, it will only show entities that match the search.

In the main view of Bluecell Alpha, when you hover over an entity it

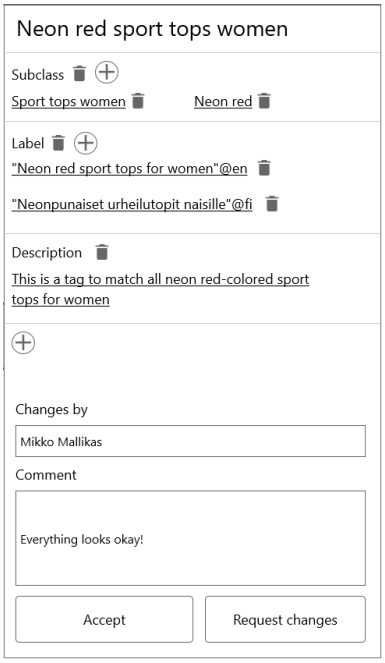

Figure 5.21: The review changes view of Bluecell Alpha.

shows a button to add a new entity, related to that one, after which it adds an empty entity to the graph that is related to the hovered one (as seen in picture 5.23). It fulfills the requirement *"I can add entities to specific parts in the graph by selecting that part"*. The point for this functionality lies in noticing in the graph view that some entity is missing and being able to add it to the correct place at once. For the sake of fast prototyping this functionality is not very detailed in the prototype.

As the main functionality for the editor is to build new ways to tag and define products in the Zalando fashion store, showing related entities, suggestions for new entities and results for the selected entity in the Zalando shop is extremely important. In the related tags and results view of Bluecell Alpha (as seen in picture 5.24) which is shown when you select a tag (as part of the edit form or create a new entity form) it first shows the related entities (entities that share similar properties or entities the selected one is related to directly), suggestions for new entities, based on the selected entity and existing ones and lastly some example results from the Zalando fashion store, such as the red sport tops in the picture. The view fulfills the requirements *"I see relevant, similar and duplicate entities when creating or editing new ones"*, *"I can see examples of search results from the Zalando*

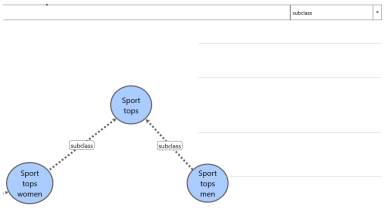

Figure 5.22: The filter view of Bluecell Alpha.

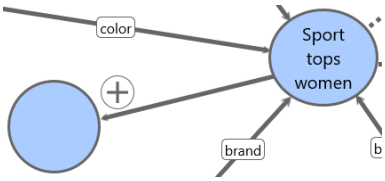

Figure 5.23: The graphical addition view of Bluecell Alpha.

shop with the selected identity" and *"I can see suggestions for new entities, based on graph analysis"*.

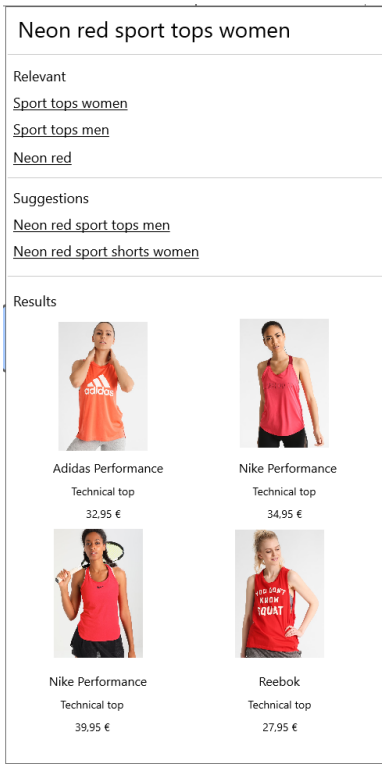

Figure 5.24: The related tags and results view of Bluecell Alpha.

5.2.2 Second prototype

Based on the second set of requirements I created another prototype using MockPlus, that was aimed at creating a visual graph based editor on top of WebVOWL and making sure all the requirements defined in the second phase are met by the prototype. The prototype itself is simply a collection of different views, but MockPlus could be used to make it into an interactive demo with limited capabilities. The visual design language is meant to follow the VOWL specification [21]. On the upper left corner of each view of the prototype are listed the requirements it fulfills, using the identification present in the first table of requirements. The code name for the tool being designed is "Bluecell" and from now onward "Bluecell Beta" will be used to designate the second prototype.

The main view of Bluecell Beta (seen in picture 5.25) shows the instances of entities in a graph-format (simplified here to save time when making the prototype). It fulfills the requirements *"I can view a visual representation of the graph"*, *"I can select the domain I want to modify (e.g. application or*

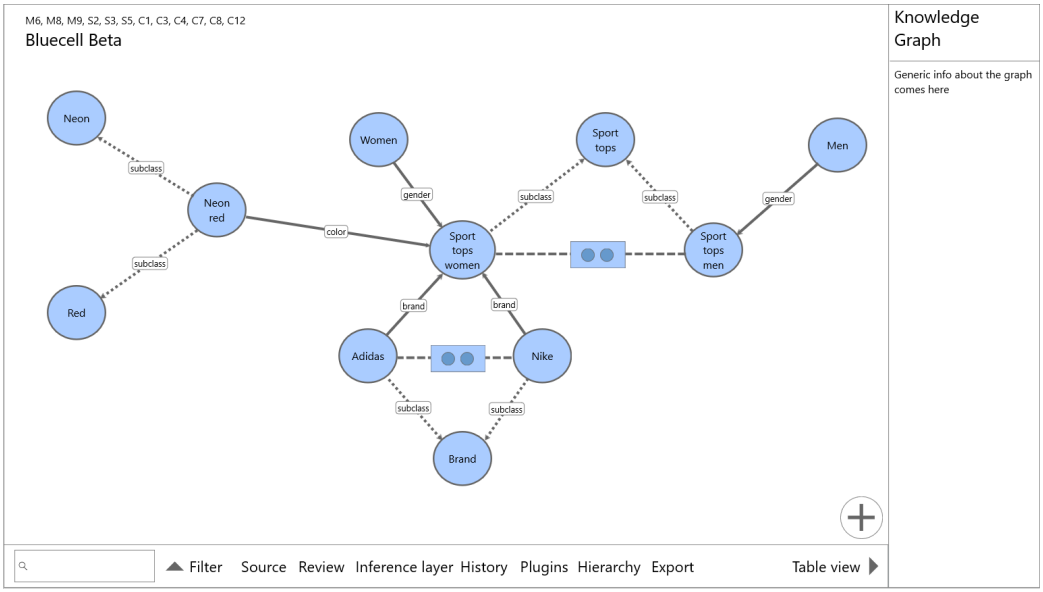

Figure 5.25: The main view of Bluecell Beta.

domain knowledge" by clicking the "Source"-button, *"I can choose between a visual representation and a tabular view of the graph"* by clicking the "Table view"-button (the table view itself can be seen in picture 5.26, *"I can see the inference layer of the graph"* by clicking the "Inference layer"-button, *"I can see the history of edits and the people who made those edits"* by clicking the "History"-button, *"I can traverse the visual representation of the graph"*, *"I can add plugins to the editor to facilitate specific tasks"* by clicking the "Plugins"-button, *"I can visualize hierarchies within the graph based on transitive predicates"* by clicking the "Hierarchy"-button, *"I can run the editor locally against any graph instance"* by clicking the "Source"-button, *"I can export the graph as JSON-LD, Turtle, RDF/XML or other similar formats"* by clicking the "Export"-button, *"I can import a graph into the editor as JSON-LD, Turtle, RDF/XML or other similar formats"* by clicking the "Source"-button and *"I can see all reviews awaiting my attention collectively"* by clicking the "Review"-button. On the bottom left part of the UI is a search bar, followed on the bottom bar with different controls for the editor. On the lower right corner is a button to create a new entity (the plus sign) and on the right side of the UI there is other information visible. Unlike the first prototype, the second prototype can influence both the entities and instances. The way to influence entities is to click the "Source"-button and select the correct ontology from the visible list.

Figure 5.26: The table view of Bluecell Beta.

After clicking the "create new instance" button, the part on the right side of the UI (as seen in picture 5.27) is populated and gives one a form with which to define the new instance. The form fulfills requirements *"I can create triples"*, *"I can edit triples"*, *"After modifying the ontology, my changes will not be visible until verified by a domain expert and an ontology engineer"* by clicking the "Submit for review"-button, *"After modifying the knowledge graph, my changes will not be visible until verified by a peer"* by clicking the "Submit for review"-button, *"IRIs for the entities I create may be human-readable"*, *"I can edit and create entities in the graph using a form with the labels of IRIs"*, *"I can see suggestions for new predicates for the entity I am currently modifying"* by expanding the "Suggestions"-tab, *"I see relevant, similar and duplicate entities when creating or editing new ones"* by expanding the "Relevant"-tab, *"I can copy an existing entity to start the creation of a new one based on it"* by clicking the "Copy"-button, *"I can see examples of search results from the Zalando shop with the selected entity (including editorial content, collections etc.)"* by expanding the "Search results"-tab, *"I can use well-defined properties and entities, such as FOAF and DBpedia"*, *"I can assign a review to a specific person or to multiple people"* by selecting the people you want from the drop-down in the "Reviewers"-tab, *"I can see suggestions for new entities, based on graph analysis"* by expanding the "Suggestions"-tab and *"I can mark an entity as "badly modelled" for future reference"* by selecting the "Badly modelled"-check-box. In the edit view the image has already been filled with the name, sub classes, labels and a description. When the sub classes and other relations have been defined it shows up in the graph. After defining the instance you can either submit it for review or cancel the changes you have made. Some details have been overlooked in the prototype in favor of clarity.

The review changes made by others view (as seen in picture 5.28) is the functionality missing from both Protégé and TopBraid Composer. The review view fulfills the requirement *"I am able to review changes to the graph made by others"*. The view shows the changes being reviewed both in the graph and in the form, with the form showing the details of the proposed changes. After reviewing changes you can type in a comment and either accept the changes or request for new changes. As per before some details have been overlooked for simplicity.

The filter view of Bluecell Beta (as seen in picture 5.29) is reached by ei-

ther selecting a property from the bottom-left drop-down list or typing something in the search field in the top. It fulfills the requirements *"I can filter and highlight the entities visible in the visual representation of the graph"* and *"I can see clusters of entities with similar properties in the visualization"*. When you select one of the properties from the drop-down menu it only shows relations of that property (as seen in the picture). When you type something into the search box, it will only show entities that match the search.

In the main view of Bluecell Alpha, when you hover over an entity it shows a button to add a new entity, related to that one, after which it adds an empty entity to the graph that is related to the hovered one (as seen in picture 5.30). It fulfills the requirement *"I can add entities to specific parts in the graph by selecting that part"*. The point for this functionality lies in noticing in the graph view that some entity is missing and being able to add it to the correct place at once. For the sake of fast prototyping this functionality is not very detailed in the prototype.

Figure 5.27: The editing and creating new view of Bluecell Beta.

Figure 5.28: The review changes view of Bluecell Beta.

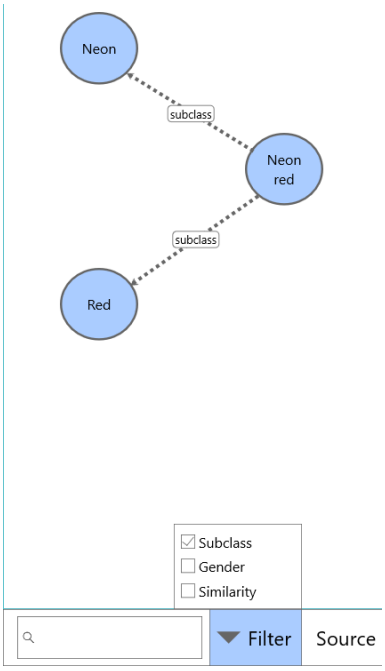

Figure 5.29: The filter view of Bluecell Beta.

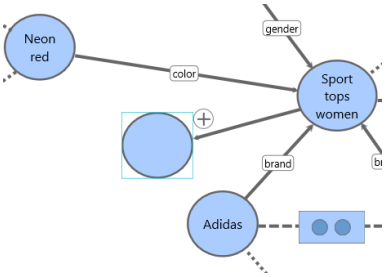

Figure 5.30: The graphical addition view of Bluecell Beta.

Chapter 6

Evaluation

Looking back on the process and how I implemented it, I believe that the process itself was good, although the actual implementation should be considered a part of the process, even if it was out of scope for this thesis. More time should have probably been spent to validate and refine the use cases given by the product owner, as the product owner did not give any input on whether or not the use cases had been discovered with end users or not. That is not a critical thing though, as the requirements were still formulated with end users and other stakeholders.

If I could redo the requirements definition phase again, I would start with filling some sort of business canvas with stakeholders, followed by workshops with end users to discover requirements and then only later have workshops with the engineering team to think more about the technical risks and related prioritization. I would have also liked to have the input of the people who gave requirements to give their validation methods, as now they were based on how I understood the requirements.

The evaluation of existing tools should have been done with a more scientific method than UX Expert evaluation, as it is dependent on the expertise of the people doing the evaluation and as I am not a designer by profession I would not place too much weight on my evaluations. For example a better way would have been to have a laboratory setting with actual end users trying to use the chosen tools. It would have also been a good idea to see if any of the existing open source editors that are no longer maintained could be used as alternatives. The property checklist was a good evaluation tool especially with WebVOWL and Protégé, but I should have bought the TopBraid Composer's commercial license to properly grade it, but I doubt it would have effected the grading by much.

During the prototyping phase it would have been good to have professional UX and visual design experts help with the actual prototyping, as

although I have some experience with tools like MockPlus I am not an expert in UX or visual design. Additionally I should have been more active in getting feedback from the end users and other stakeholders about the prototypes and especially their usability, but I hope when the implementation begins, iterative methods will be followed. Also more iterations should have been done, but beginning the implementation of the technically most difficult features after the first one or two prototypes and then iterating with the actual product.

Chapter 7

Conclusions

The work started with 2 workshops to define a set of requirements for an ontology management tool to be used by domain experts inside Zalando. After the requirements had been defined and refined, I evaluated the existing ontology management tools with the UX expert evaluation and the property checklist (using the defined requirements) and came to the conclusion that none of the three options meet all of the necessary requirements. As that was the case I created two prototypes of a possible solution, called Bluecell Alpha and Bluecell Beta, of which Beta builds on top of WebVOWL. Doing the same set of evaluation methodologies for the prototypes leads to the conclusion that the only way to meet the set of requirements would be to implement Bluecell Beta.

The existing ontology management tools are clearly aimed at ontology professionals and not for domain experts who do not have the necessary technical expertise to read and write triples. If domain experts were not the target group, I would suggest using Protégé, as it is highly extensible, open sourced and very well documented. However domain experts are the target group in this case and that is why my proposal would be to build upon WebVOWL to create an ontology management tool aimed especially at domain experts, as it has the most intuitive user interface of the tools reviewed. My suggestion would be building Bluecell Beta (the second prototype) on top of WebVOWL as it has been designed especially for this use case. Building the editor as open source might also have a positive impact on the employee branding of Zalando, as open source projects are well respected in software development. Also as semantic web is still a niche in any industry, allocating significant resources into the subject could help Zalando set the future of semantic web in the fashion world.

Bibliography

[1] Antoniou, G., and Van Harmelen, F. *A semantic web primer*. MIT press, 2004.

[2] Berners-Lee, T. Linked data-design issues. *http://www. w3. org/DesignIssues/LinkedData. html* (2006).

[3] Brennan, K., et al. *A Guide to the Business Analysis Body of Knowledger*. Iiba, 2009.

[4] Brickley, D., Guha, R. V., and McBride, B. Rdf schema 1.1. *W3C recommendation 25* (2014), 2004–2014.

[5] Brickley, D., and Miller, L. Foaf vocabulary specification 0.91, 2007.

[6] Clegg, D., and Barker, R. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc., 1994.

[7] Cordes, A. 5 popular requirements prioritisation techniques, Jul 2014.

[8] Dick, J., Hull, E., and Jackson, K. *Requirements engineering*. Springer, 2017.

[9] Dürst, M., and Suignard, M. Internationalized resource identifiers (iris). Tech. rep., 2004.

[10] Google. Knowledge – inside search – google, 2017.

[11] Guha, R. V., Brickley, D., and Macbeth, S. Schema. org: Evolution of structured data on the web. *Communications of the ACM 59*, 2 (2016), 44–51.

[12] Heath, T., and Bizer, C. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology 1*, 1 (2011), 1–136.

[13] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S. Owl 2 web ontology language primer. *W3C recommendation 27*, 1 (2009), 123.

[14] Initiative, D. C. M., et al. Dublin core metadata element set, version 1.1.

[15] Jordan, P. W. *Designing pleasurable products: An introduction to the new human factors*. CRC press, 2002.

[16] Lassila, O., Swick, R. R., et al. Resource description framework (rdf) model and syntax specification.

[17] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web 6*, 2 (2015), 167–195.

[18] Lohmann, S., Negru, S., Haag, F., and Ertl, T. Visualizing ontologies with vowl. *Semantic Web 7*, 4 (2016), 399–419.

[19] Masinter, L., Berners-Lee, T., and Fielding, R. T. Uniform resource identifier (uri): Generic syntax.

[20] Musen, M. A. The protégé project: a look back and a look forward. *AI matters 1*, 4 (2015), 4–12.

[21] Negru, S., Lohmann, S., Haag, F., et al. Vowl: Visual notation for owl ontologies, 2014.

[22] Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. A design science research methodology for information systems research. *Journal of management information systems 24*, 3 (2007), 45–77.

[23] Ricadela, A. Zalando to raise up to $815 million in german e-commerce ipo, Sep 2014.

[24] Scott, M. In shadow of amazon, european challenger looks to china for inspiration, Dec 2017.

[25] Väänänen-Vainio-Mattila, K., and Wäljas, M. Developing an expert evaluation method for user experience of cross-platform web services. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era* (2009), ACM, pp. 162–169.

[26] Weiss, R. Zalando slumps as european retailers battle for online share, Apr 2017.

Appendix A

Workshop results

Table A.1: Results of the first workshop

ID	Requirement	Source	Priority	Graph	Verification
M1	I can create triples	Workshop 10.7.2017	Must	Both	After creating triples in the service, they will be present in the graph
M2	I can edit triples	Workshop 10.7.2017	Must	Both	After editing triples in the service, they will be present in the graph

M3	After modifying the ontology, my changes will not be visible until verified by a domain expert and an ontology engineer	Workshop 10.7.2017	Must	Ontology	After modifying the ontology, changes will not be present in the graph until they have been reviewed by at least a domain expert and an ontology engineer
M4	After modifying the knowledge graph, my changes will not be visible until verified by a peer	Workshop 10.7.2017	Must	Knowledge graph	After modifying the knowledge graph, changes will not be present in the graph until they have been reviewed by at least 1 peer
M5	I am able to review changes to the graph made by others	Workshop 10.7.2017	Must	Both	After someone else makes changes to the graph, there is a way to review them before they are applied

M6	IRIs for the entities I create may be human-readable	Workshop 10.7.2017	Must	Both	IRIs within the graph are human-readable (IRIs have a meaning)
M7	IRIs within the graph are human-readable (IRIs have a meaning)	Workshop 10.7.2017	Must	Both	There is a visual form with labels for editing and creating entities in the graph
M8	I can view a visual representation of the graph	Workshop 10.7.2017	Must	Both	There is a visual representation of the graph
S1	I can use well-defined properties and entities, such as FOAF and DBpedia	Workshop 10.7.2017	Should	Both	There is a way to use properties and entities from external ontologies, such as FOAF and DBpedia
S2	I can filter the entities visible in the visual representation of the graph	Workshop 10.7.2017	Should	Both	There is a way to filter what entities are visible in the graph representation

S3	I can traverse the visual representation of the graph	Workshop 10.7.2017	Should	Both	There is a way to explore different parts of the graph using the visual representation
S4	I see relevant, similar and duplicate entities when creating or editing new ones	Workshop 10.7.2017	Should	Both	There are relevant, similar and duplicate entities visible within the form for editing and creating
S5	I can visualize hierarchies within the graph based on transitive predicates	Workshop 10.7.2017	Should	Both	There is a way to visualize hierarchies within the graph by selecting different predicates
C1	I can see deduced predicates in entities	Workshop 10.7.2017	Could	Both	There are deduced predicates visible when viewing entities (for example based on transitive properties)

C2	I can add entities to specific parts in the graph by selecting that part	Workshop 10.7.2017	Could	Both	There is a way to select a position in the graph and add an entity to that position
C3	I can see examples of search results from the Zalando shop with the selected entity	Workshop 10.7.2017	Could	Both	When viewing an entity in the knowledge graph, there are examples of search results visible
C4	I can see suggestions for new entities, based on graph analysis	Workshop 10.7.2017	Could	Both	There are suggestions for new entities visible, based on graph analysis
C5	I can export the graph as a CSV or a JSON	Workshop 10.7.2017	Could	Both	There is a way to export the graph as a single file
W1	The editor does not use too many resources on the backend services	Workshop 10.7.2017	Won't	Both	The graph editor does not tax the backend services too much

W2	I can create new entities in batches	Workshop 10.7.2017	Won't	Both	There is a way to create multiple entities at the same time
W3	I want to type search phrases into the editor and receive appropriate entities as a result	Workshop 10.7.2017	Won't	Both	I want to type search phrases into the editor and receive appropriate entities as a result
W4	I want to be able to perform semantic-based searches against the graph and see the appropriate visualizations	Workshop 10.7.2017	Won't	Both	There is a way to perform semantic-based searches on the graph and a visualization of the results

Table A.2: Results of the second workshop

ID	**Old ID**	**Requirement**	**Source**	**Priority**	**Verification**
M1	M1	I can create triples	Workshop 10.7.2017	Must	After creating triples in the service, they will be present in the graph

M2	M2	I can edit triples	Workshop 10.7.2017	Must	After editing triples in the service, they will be present in the graph
M3	M3	After modifying the ontology, my changes will not be visible until verified by a domain expert and an ontology engineer	Workshop 10.7.2017	Must	After modifying the ontology, changes will not be present in the graph until they have been reviewed by at least a domain expert and an ontology engineer
M4	M4	After modifying the knowledge graph, my changes will not be visible until verified by a peer	Workshop 10.7.2017	Must	After modifying the knowledge graph, changes will not be present in the graph until they have been reviewed by at least 1 peer

M5	M5	I am able to review changes to the graph made by others	Workshop 10.7.2017	Must	After someone else makes changes to the graph, there is a way to review them before they are applied
M6	M8	I can view a visual representation of the graph	Workshop 10.7.2017	Must	There is a visual representation of the graph
M7	S2	I can filter and highlight the entities visible in the visual representation of the graph	Workshop 10.7.2017 Updated 26.9.2017	Must	There is a way to filter what entities are visible in the graph representation and interesting ones are highlighted
M8	-	I can select the domain I want to modify (e.g. application or domain knowledge)	Workshop 26.9.2017	Must	There is a way to select the domain that is being modified

M9	-	I can choose between a visual representation and a tabular view of the graph	Workshop 26.9.2017	Must	There are both a visual representation and a tabular view and there is a way to change between them
M10	M6	IRIs for the entities I create may be human-readable	Workshop 10.7.2017 New priority 26.9.2017	Must	IRIs within the graph are human-readable (IRIs have a meaning)
S1	M7	I can edit and create entities in the graph using a form with the labels of IRIs	Workshop 10.7.2017 New priority 26.9.2017	Should	There is a visual form with labels for editing and creating entities in the graph
S2	C1	I can see the inference layer of the graph	Workshop 10.7.2017 Updated 26.9.2017	Should	There is a way to view the inference layer of the ontology
S3	-	I can see the history of edits and the people who made those edits	Workshop 26.9.2017	Should	There is a way to view the history of edits and the people who made those edits

S4	-	I can see suggestions for new predicates for the entity I am currently modifying	Workshop 26.9.2017	Should	There is a way to see suggestions for new predicates based on for example inference
S5	S3	I can traverse the visual representation of the graph	Workshop 10.7.2017 New priority 26.9.2017	Should	There is a way to explore different parts of the graph using the visual representation
S6	S4	I see relevant, similar and duplicate entities when creating or editing new ones	Workshop 10.7.2017 New priority 26.9.2017	Should	There are relevant, similar and duplicate entities visible within the form for editing and creating
S7	-	I can copy an existing entity to start the creation of a new one based on it	Workshop 26.9.2017	Should	There is a way to start the creation of a new entity based on an existing entity

S8	C3	I can see examples of search results from the Zalando shop with the selected entity (including editorial content, collections etc.)	Workshop 10.7.2017 Updated 26.9.2017	Should	When viewing an entity in the knowledge graph, there are examples of search results, editorial content, collections etc. visible
C1	-	I can add plugins to the editor to facilitate specific tasks	Workshop 26.9.2017	Could	There is a way to add plugins to the editor to facilitate specific tasks
C2	S1	I can use well-defined properties and entities, such as FOAF and DBpedia	Workshop 10.7.2017 New priority 26.9.2017	Could	There is a way to use properties and entities from external ontologies, such as FOAF and DBpedia
C3	S5	I can visualize hierarchies within the graph based on transitive predicates	Workshop 10.7.2017 New priority 26.9.2017	Could	There is a way to visualize hierarchies within the graph by selecting different predicates

C4	-	I can run the editor locally against any graph instance	Workshop 26.9.2017	Could	There is a way to run the editor locally against any graph instance
C5	-	I can assign a review to a specific person or to multiple people	Workshop 26.9.2017	Could	There is a way to assign reviews to specific people
C6	-	I can see clusters of entities with similar properties in the visualization	Workshop 10.7.2017 Updated 26.9.2017	Could	There is a way to see clusters of entities with similar properties in the visualization
C7	C5	I can export the graph as JSON-LD, Turtle, RDF/XML or other similar formats	Workshop 26.9.2017	Could	There is a way to export the graph as a single file in multiple semantic formats
C8	-	I can import a graph into the editor as JSON-LD, Turtle, RDF/XML or other similar formats	Workshop 26.9.2017	Could	I can import a graph into the editor as JSON-LD, Turtle, RDF/XML or other similar formats

C9	C2	I can add entities to specific parts in the graph by selecting that part	Workshop 10.7.2017 New priority 26.9.2017	Could	There is a way to select a position in the graph and add an entity to that position
C10	C4	I can see suggestions for new entities, based on graph analysis	Workshop 10.7.2017 New priority 26.9.2017	Could	There are suggestions for new entities visible, based on graph analysis
C11	-	I can mark an entity as "badly modelled" for future reference	Workshop 26.9.2017	Could	There is a way to mark an entity as badly modelled
C12	-	I can see all reviews awaiting my attention collectively	Workshop 26.9.2017	Could	There is a way to see all reviews awaiting review in the editor
W1	W1	The editor does not use too many resources on the backend services	Workshop 10.7.2017	Won't	The graph editor does not tax the backend services too much

W2	W2	I can create new entities in batches	Workshop 10.7.2017	Won't	There is a way to create multiple entities at the same time
W3	W3	I want to type search phrases into the editor and receive appropriate entities as a result	Workshop 10.7.2017	Won't	There is some form of search bar that returns entities from the graph as results
W4	W4	I want to be able to perform semantic-based searches against the graph and see the appropriate visualizations	Workshop 10.7.2017	Won't	There is a way to perform semantic-based searches on the graph and a visualization of the results
W5	-	I can focus on 1 level of the graph at a time	Workshop 26.9.2017	Won't	There is a way to focus on a single level of the graph at a time
W6	-	The editor has default filters when first opened	Workshop 26.9.2017	Won't	There are default filters applied to the editor at start

W7	-	The editor has a mobile interface or application	Workshop 26.9.2017	Won't	There is a mobile interface or application for the editor
W8	-	I can set a reminder date for when something needs to be reviewed again	Workshop 26.9.2017	Won't	There is a way to set reminder dates for reviewing something again

Appendix B

Requirements categorization

Table B.1: Requirements categorization

ID	**Requirement**	**Topic**
M1	I can create triples	Management
M2	I can edit triples	Management
M3	After modifying the ontology, my changes will not be visible until verified by a domain expert and an ontology engineer	Review
M4	After modifying the knowledge graph, my changes will not be visible until verified by a peer	Review
M5	I am able to review changes to the graph made by others	Review
M6	I can view a visual representation of the graph	Representation
M7	I can filter and highlight the entities visible in the visual representation of the graph	Representation
M8	I can select the domain I want to modify (e.g. application or domain knowledge)	Management
M9	I can choose between a visual representation and a tabular view of the graph	Representation

M10	IRIs for the entities I create may be human-readable	Management
S1	I can edit and create entities in the graph using a form with the labels of IRIs	Representation
S2	I can see the inference layer of the graph	Ontological capabilities
S3	I can see the history of edits and the people who made those edits	Review
S4	I can see suggestions for new predicates for the entity I am currently modifying	Suggestions
S5	I can traverse the visual representation of the graph	Representation
S6	I see relevant, similar and duplicate entities when creating or editing new ones	Suggestions
S7	I can copy an existing entity to start the creation of a new one based on it	Management
S8	I can see examples of search results from the Zalando shop with the selected entity (including editorial content, collections etc.)	Suggestions
C1	I can add plugins to the editor to facilitate specific tasks	Extensibility
C2	I can use well-defined properties and entities, such as FOAF and DBpedia	Ontological capabilities
C3	I can visualize hierarchies within the graph based on transitive predicates	Ontological capabilities
C4	I can run the editor locally against any graph instance	Extensibility
C5	I can assign a review to a specific person or to multiple people	Review
C6	I can see clusters of entities with similar properties in the visualization	Representation
C7	I can export the graph as JSON-LD, Turtle, RDF/XML or other similar formats	Management

C8	I can import a graph into the editor as JSON-LD, Turtle, RDF/XML or other similar formats	Extensibility
C9	I can add entities to specific parts in the graph by selecting that part	Management
C10	I can see suggestions for new entities, based on graph analysis	Suggestions
C11	I can mark an entity as "badly modelled" for future reference	Review
C12	I can see all reviews awaiting my attention collectively	Review

Appendix C

Evaluation of existing tools

Table C.1: Evaluation of WebVOWL

ID	Implements	Points
M1	**No**, possible by extending the source code	2
M2	**No**, possible by extending the source code	2
M3	**No**, possible by extending the source code	2
M4	**No**, possible by extending the source code	2
M5	**No**, possible by extending the source code	2
M6	**No**, possible by extending the source code	2
M7	**Yes**	4
M8	**Yes**, to a limit, possible to improve by extending the source code	2
M9	**No**, possible by extending the source code	2
M10	**No**, possible by extending the source code	2
S1	**No**, possible by extending the source code	1

S2	**No**, possible by extending the source code	1
S3	**No**, possible by extending the source code	1
S4	**No**, possible by extending the source code	1
S5	**Yes**	2
S6	**No**, possible by extending the source code	1
S7	**No**, possible by extending the source code	1
S8	**No**, possible by extending the source code	1
C1	**Yes**, by extending the source code	½
C2	**Yes**	1
C3	**No**, possible by extending the source code	½
C4	**Yes**	1
C5	**No**, possible by extending the source code	½
C6	**Yes**	1
C7	**Yes**, to a limit, possible to improve by extending the source code	½
C8	**Yes**, to a limit, possible to improve by extending the source code	½
C9	**No**, possible by extending the source code	½
C10	**No**, possible by extending the source code	½
C11	**No**, possible by extending the source code	½
C12	**No**, possible by extending the source code	½

Table C.2: Evaluation of Protégé

ID	Implements	Points
M1	**Yes**	4
M2	**Yes**	4
M3	**No**, possible by creating a plugin	2
M4	**No**, possible by creating a plugin	2
M5	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Project_Management`	2
M6	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Visualization`	2
M7	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Visualization`	2
M8	**Yes**	4
M9	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Visualization`	2
M10	**Yes**	4
S1	**Yes**	2
S2	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Inference`	1
S3	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Project_Management`	1
S4	**No**, possible by creating a plugin	1
S5	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Visualization`	1
S6	**No**, possible by creating a plugin	1
S7	**No**, possible by creating a plugin	1
S8	**No**, possible by creating a plugin	1

C1	**Yes**	1
C2	**Yes**	1
C3	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Visualization`	½
C4	**Yes**	1
C5	**No**, possible by creating a plugin	½
C6	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Visualization`	½
C7	**Yes**	1
C8	**Yes**	1
C9	**No**, possible by creating a plugin	½
C10	**No**, possible by creating a plugin	½
C11	**Via plugin**, several possibilities `https://protegewiki.stanford.edu/wiki/Project_Management`	½
C12	**No**, possible by creating a plugin	½

Table C.3: Evaluation of TopBraid Composer

ID	**Implements**	**Points**
M1	**Yes**	4
M2	**Yes**	4
M3	**No**	0
M4	**No**	0
M5	**No**	0

M6	**Yes**, but it requires a commercial license	2
M7	**Maybe**, but it requires a commercial license	2
M8	**Maybe**, but it requires a commercial license	2
M9	**Yes**, but it requires a commercial license	2
M10	**Yes**	4
S1	**Yes**	2
S2	**No**	0
S3	**No**	0
S4	**No**	0
S5	**Yes**, but it requires a commercial license	1
S6	**No**	0
S7	**No**	0
S8	**No**	0
C1	**No**	0
C2	**Yes**	1
C3	**Maybe**, but it requires a commercial license	½
C4	**Yes**	1
C5	**No**	0
C6	**Maybe**, but it requires a commercial license	½

C7	**Yes**	1
C8	**Yes**	1
C9	**Maybe**, but it requires a commercial license	½
C10	**No**	0
C11	**No**	0
C12	**No**	0